IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

System and Method for Performing Garbage Collection on a Large Heap

Inventor:
Patrick H. Dussud

ATTORNEY'S DOCKET NO. MS1-1962US

TECHNICAL FIELD

[0001] This document generally relates to the management of memory in a computer system, and more particularly, to the management of memory in support of garbage collection.

BACKGROUND

[0002] Garbage collection, also commonly referred to as automatic memory management, attempts to automatically recycle dynamically allocated memory. Dynamically allocated memory is memory that is created at runtime and that is stored in an area of memory commonly referred to as the heap. There are several techniques that have been developed to perform garbage collection.

[0003] One technique is full garbage collection. During full garbage collection, the entire heap is analyzed to determine which memory to recycle. A disadvantage with this technique is that as the heap becomes larger and larger, garbage collection causes a significant delay to programs that are executing. This delay prohibits utilizing full garbage collection when the heap becomes too large.

[0004] Another type of garbage collection, commonly referred to as an incremental, generational, or ephemeral garbage collection, attempts to overcome this execution delay by dividing the heap into two or more generations. Newly created objects are allocated in the "youngest" generation. Ephemeral garbage collection then analyzes the "youngest" generation frequently in order to collect and recycle these objects. The older generations are analyzed less frequently, and, when recycled, all of the objects in the older generation are recycled together. Objects in the "youngest" generation that survive a certain number of collections may be "promoted" to the next

older generation. Because the ephemeral garbage collection is typically performed on a small portion of the entire heap, ephemeral garbage collection does not impact the execution of programs in a significant manner.

[0005] However, even though ephemeral garbage collection does not directly impact the execution of programs in a significant manner, it does cause other problems. One problem occurs when older generations contain pointers that reference younger generations. Without scanning all the older generations, objects in the younger generation may be erroneously recycled. However, if all the generations are scanned, then the benefits of having multiple generations disappear. Fortunately, techniques have been developed to overcome this problem.

[0006] One technique, commonly referred to as card marking, divides the heap into cards of equal size. The size of the card may vary, but it is typically bigger than a word and smaller than a page. There are various acceptable methods of marking objects. For example, a single bit within a card bitmap can be changed to indicate when the memory associated with the card has been "touched" or accessed (e.g., written to). Thus, when performing ephemeral garbage collection, the objects in the youngest generation and the objects in each of the cards in the older generations that are indicated as being written into are analyzed. The term "written card" is used through-out this document to refer to a card that contains memory locations that have been "touched". While card marking greatly reduces the amount of heap that is analyzed during garbage collection, the efficiency of card marking is dependent on the size of the heap. For example, if the card size becomes too large, the cost of analyzing each of the objects in each of the marked cards becomes prohibitive. On the other hand, if the card size is too small and there are numerous cards, the overhead of having so many cards becomes prohibitive.

Thus, the benefit of ephemeral garbage collection employing card marking also decreases as the size of the heap increases.

[0007] Recently, a new technique has emerged that helps minimize the dependency of ephemeral garbage collection on the heap size. This new technique implements a hierarchy of bundles where each bundle is associated with multiple cards. A bundle bit map is employed where each bit represents one of the bundles. During garbage collection, the bundle bit map is checked to determine which bundles have objects that have been accessed. If the bundle bit map indicates that an object within the bundle has been accessed, each card in that bundle is checked to see if it has been accessed. If it has, then each of its objects is checked. While this technique improves the efficiency of the ephemeral garbage collection, the cost of executing the program is doubled. For example, turning to FIG. 1, pseudo-code 100 illustrating a portion of helper code 102 called by a compiler is shown. The helper code 102 is called whenever a store operation (e.g., "=" operator) is encountered in the program that is being executed. A first statement 104 performs the necessary work of storing a value into the location specified in the program. The second statement 106 performs overhead for marking the card associated with the location, assuming the location is not in the "youngest" generation. The third statement 108 performs overhead for marking the bundle associated with the location. Therefore, the addition of statement 106 to perform bundling doubles the overhead for executing the store operator 104 in comparison to only performing card-marking (statement 106). Because programs typically contains several store operations, which are commonly within a loop, this doubling of execution for each store operation becomes unacceptable and implementing bundles becomes prohibitive.

[0008] Thus, until now, there has not been a satisfactory solution for ephemeral garbage collection of a large heap.

SUMMARY

[0009] The techniques and mechanisms described herein are directed to a system for performing garbage collection on a large heap. The heap is divided into cards, which are grouped into bundles. Briefly stated, the techniques include initiating a write-watch mechanism to track accesses to specified cards. The write-watch mechanism provides a list of the written cards to a garbage collection process which determines marked (accessed) bundles based on the list. For each marked bundle, the marked cards within the marked bundle are scanned to identify the accessed objects. The accessed objects are then collected. Because determining the marked bundles is performed at the start of the garbage collection process and not whenever the cards within the bundle are accessed, the present technique reduces the overhead associated with bundle marking and allows the efficiency of the garbage collection process to be less dependent on heap size.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] Non-limiting and non-exhaustive embodiments are described with reference to the following figures, wherein like reference numerals refer to like parts throughout the various views unless otherwise specified.

[0011] FIG. 1 is pseudo-code illustrating a prior technique of managing memory for garbage collection.

[0012] FIG. 2 is an illustrative operating environment suitable for implementing the techniques and mechanisms described herein.

[0013] FIG. 3 is an illustrative computer environment that may be used to implement the techniques and mechanisms described herein.

[0014] FIG. 4 is a block diagram that further illustrates aspects of the system memory shown in FIG. 3 for implementing the techniques and mechanisms described herein.

[0015] FIG. 5 is a flow diagram illustrating a portion of the execution process that utilizes memory management features that supports garbage collection.

[0016] FIG. 6 is a flow diagram illustrating an ephemeral garbage collection process.

DETAILED DESCRIPTION

[0017] Briefly, the present system and method minimize the overhead required in implementing bundle and card marking in ephemeral garbage collection. This is achieved by utilizing a memory management feature known as "write-watch" that is responsible for tracking modifications to specified memory locations. The "write-watch" information is used by the ephemeral garbage collection process to determine which bundles in the older generations have objects that need to be collected. The "write-watch" mechanism tracks the first access to a memory location and does not track subsequent accesses to the same memory location. As will be described in detail below, the present ephemeral garbage collection process allows the program to execute more efficiently without adding unnecessary overhead. These and other advantages will become clear after reading the following detailed description.

EXEMPLARY OPERATING ENVIRONMENT

[0018] FIG. 2 is an illustrative operating environment 200 suitable for implementing the techniques and mechanisms described herein. Operating environment 200 includes source 202 that is input into compiler 204. The source 202 may take several formats, but is typically a text-based file written in some language (e.g., C#, C++, Visual Basic, HTML). Compiler 204 compiles the source 202 to create application 206. Again, application 206 may take several forms, such as an executable that does not need further compilation or an assembly that may require additional compilation.

[0019] During runtime, application 206 is executed by utilizing framework 208, runtime environment 210, and operating system 212. Framework 208 may be a set of libraries or other services. Runtime environment is responsible for performing many services, such as encryption, security, Just-in-Time (JIT) compilation, and others. One service pertinent to the present technique is garbage collection 220. Briefly, garbage collection 220, described below in conjunction with the flow diagram in FIG. 6, performs ephemeral garbage collection. As will be described below, the ephemeral garbage collection 220 utilizes information obtained from a memory manager 222. As shown, memory manager 222 may be part of operating system 212. However, memory manager 222 may operate within runtime environment 210 or be its own application 206. In any implementation, memory manager 222 provides a "write-watch" mechanism that identifies the accessing of memory a first time without performing redundant identification of activities each time the memory is accessed. This "write-watch" mechanism may be performed using hardware, software, or a combination of both. Although any implementation of a "write-watch" mechanism may be used, the techniques described here have been used in conjunction with a "write-watch" mechanism which

serves as the subject of U.S. Patent Application Ser. No. 09/628708, entitled "EFFICIENT WRITE-WATCH MECHANISM USEFUL FOR GARBAGE COLLECTION IN A COMPUTER," filed on July 31, 2000, and expressly incorporated herein by reference for all purposes.

EXEMPLARY COMPUTING ENVIRONMENT

[0020] The various embodiments of the present ephemeral garbage collection may be implemented in different computer environments. The computer environment shown in FIG. 3 is only one example of a computer environment and is not intended to suggest any limitation as to the scope of use or functionality of the computer and network architectures. Neither should the computer environment be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the example computer environment.

[0021] With reference to FIG. 3, one exemplary system for implementing the present ephemeral garbage collection includes a computing device, such as computing device 300. In a very basic configuration, computing device 300 typically includes at least one processing unit 302 and system memory 304. Depending on the exact configuration and type of computing device, system memory 304 may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.) or some combination of the two. System memory 304 typically includes an operating system 305, one or more program modules 306, and may include program data 307. This basic configuration is illustrated in FIG. 3 by those components within dashed line 308.

[0022] Computing device 300 may have additional features or functionality. For example, computing device 300 may also include additional data storage devices (removable and/or non-removable) such as, for example, magnetic disks, optical disks, or

tape. Such additional storage is illustrated in FIG. 3 by removable storage 309 and non-removable storage 310. Computer storage media may include volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information, such as computer readable instructions, data structures, program modules, or other data. System memory 304, removable storage 309 and non-removable storage 310 are all examples of computer storage media. Thus, computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by computing device 300. Any such computer storage media may be part of device 300. Computing device 300 may also have input device(s) 312 such as keyboard, mouse, pen, voice input device, touch input device, etc. Output device(s) 314 such as a display, speakers, printer, etc. may also be included. These devices are well know in the art and need not be discussed at length here.

[0023] Computing device 300 may also contain communication connections 316 that allow the device to communicate with other computing devices 318, such as over a network. Communication connection(s) 316 is one example of communication media. Communication media may typically be embodied by computer readable instructions, data structures, program modules, or other data in a modulated data signal, such as a carrier wave or other transport mechanism, and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such

as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. The term computer readable media as used herein includes both storage media and communication media.

[0024] Various modules and techniques may be described herein in the general context of computer-executable instructions, such as program modules, executed by one or more computers or other devices. Generally, program modules include routines, programs, objects, components, data structures, etc. for performing particular tasks or implement particular abstract data types. These program modules and the like may be executed as native code or may be downloaded and executed, such as in a virtual machine or other just-in-time compilation execution environment. Typically, the functionality of the program modules may be combined or distributed as desired in various embodiments.

[0025] An implementation of these modules and techniques may be stored on or transmitted across some form of computer readable media. Computer readable media can be any available media that can be accessed by a computer. By way of example, and not limitation, computer readable media may comprise "computer storage media" and "communications media."

[0026] FIG. 4 is a block diagram that further illustrates aspects of the system memory 304 shown in FIG. 3 for implementing the techniques and mechanisms described herein. As illustrated in FIG. 3 above, the system memory 304 may include portions of operating system 305, program modules 306, and program data 307. The operating system 305 may include a memory manager 222 that implements a write-watch mechanism that supports the present ephemeral garbage collection process. Alternatively, the memory manager 222 may be included with runtime environment 210 or be its own application 206. The program modules 306 include each application 206

that is executing, framework 208, and runtime environment 210. As mentioned above, the runtime environment 210 provides additional services and may be a cross-platform run-time environment. One of the services provided by the runtime environment 210 that is of interest in the present discussion is garbage collector 220.

[0027] The garbage collector 222 is responsible for reclaiming memory from memory heap 402. The memory heap 402 includes memory allocated for static objects, global objects, local variables, pointers, and such for each application 206 that is executing or that has executed. In addition, the memory heap 402 includes objects allocated for the operating system and the runtime environment. Typically, the objects allocated for the operating system and the runtime environment are considered long-lived objects and are not necessarily collected during the ephemeral garbage collection.

[0028] In order to separate long-lived objects from the short-lived objects, the memory heap 402 is divided into one or more generations (e.g., generations 410-416). The size of each generation 410-416 is configurable and is, typically, a contiguous range of bytes in the system memory 304. However, the size of each generation may not be identical. For example, because the ephemeral generation (e.g., generation 410) contains short lived objects which are garbage collected more frequently, the ephemeral generation may be considerably smaller than older generations (e.g., generations 410-416), which hold long-lived objects. Each of the generations are further divided into a pre-determined number of cards (e.g., cards 420-434), where the card size is typically the same and range from a byte to a page in size. A pre-determined number of cards (e.g., 420-424) are grouped into a plurality of bundles (e.g., bundle 440). While FIG. 4, illustrates bundle 440 including each of the cards 420-424 within the first generation 410, each generation may have multiple bundles.

[0029] In order to aid in the ephemeral garbage collection process, the program data 307 also includes a card table 404. The card table identifies which cards in the generations 410-416 have an object that has been accessed (e.g., written to). When there are multiple generations (e.g., generations 412-416), there may be one card table for each generation or one card table may handle all the cards for all the generations. In one implementation, the card table may be a bit map having one bit for each card. The bit for each card indicates which cards are associated with an object that has been accessed. The bit may indicate this by being set or by being clear. In other implementation, multiple bits may be used to indicate the status of each card. As shown in FIG. 2 and explained above, during compilation of the program code, an overhead statement (e.g., statement 106) is performed to set/clear the card associated with the specified location (loc). While the pseudo-code illustrates one line of code to perform this operation, those skilled in the art appreciate that numerous operations must occur in order for this to be performed. For example, the location must be associated with a specific card, the corresponding bit map must be determined, a single bit within the bitmap must be written, and the like. All of these operations contribute to overhead associated with the store statement.

[0030] The program data 307 also includes a bundle table 406 and write-watch information 408. The write-watch information 408 contains information describing the cards that have been written to since the garbage collection process was last performed. The bundle table 406 maintains information for each bundle (e.g., bundle 430). There may be one bundle table for the entire heap, multiple bundle tables based on the number of generations, multiple bundle tables based on the heap size, or the like. In one implementation, the size of memory associated with each bundle is a page of card table memory. As will be explained later in conjunction with FIGS. 6 and 7, the bundle table

406 is updated when the ephemeral garbage collection process is executed. It is updated based on the write-watch information 408 that is maintained by the write-watch mechanism in the memory manager 222. In contrast with prior attempts with implementing bundles, the present ephemeral garbage collection does not introduce the doubling of overhead (e.g., statement 108 shown in FIG. 1) when maintaining the bundle table 406. Thus, the present ephemeral garbage collection process scales to accommodate large heap sizes without adversely impacting the execution time of programs.

[0031] FIG. 5 is a flow diagram illustrating a portion of a program execution process that supports the present ephemeral garbage collection. At block 502, a mechanism is set to watch the memory pages of interest. This may involve calling a function in the memory manager to enable the write-watch mechanism for the specified memory pages. For the present ephemeral garbage collection, the specified memory pages include the memory locations storing the card table 404 shown in FIG. 4. In one implementation, the garbage collection process may supply a range of memory addresses to the write-watch mechanism for the mechanism to track. In another implementation, when there are multiple card tables, the garbage collection process may call the write-watch mechanism to watch each card table independently. Alternatively, the garbage collection process may call the write-watch mechanism to watch any number or any portion of card tables(s). Once this mechanism is set, the execution of the program may proceed.

[0032] At block 504, each statement in the program is processed for execution. If the statement is executable code, the statements do not need additional compilation. However, if the program is an assembly, a JIT compiler may need to perform additional

compilation on the statements before the statements are considered executable statements. Processing continues at decision block 506.

[0033] At decision block 506, a determination is made whether the statement includes a store operator, such as an equal ("="). If the statement does not have a store operator, the statement is executed at block 508 using processing known to those skilled in the art. Otherwise, processing continues at block 510.

[0034] At block 510, a value specified in the statement is stored at a location specified in the statement. Thus, the value is stored somewhere in the memory heap. Processing continues at decision block 512.

[0035] At decision block 512, a determination is made whether the location is within the ephemeral generation. Because locations within the ephemeral generation are already collected during the ephemeral garbage collection, additional processing is not necessary. If the location is within the ephemeral generation, processing continues to block 504 where the next statement is retrieved. Otherwise, processing continues at block 514.

[0036] At block 514, a card associated with the location is set. This may involve setting a bit in the card bitmap table associated with the location. As mentioned above, the processing performed in block 514 is considered overhead for executing the store operator. Thus, the execution of the program will incur a delay in execution. After the card is set (or cleared) in block 514, processing then continues to block 504 until each of the statements in the program have been processed.

[0037] Of interest, one will note that even though the present ephemeral garbage collection process utilizes both bundles and cards, the bundles are not set while processing each statement of the program. Thus, the additional overhead statement (e.g.,

statement 108 in FIG. 1) of setting the bundle table is eliminated. By eliminating the setting of the bundle table, the present ephemeral garbage collection process executes the program in nearly the same speed as when garbage collection utilizes only card marking. However, by utilizing the write-watch mechanism, the present ephemeral garbage collection process operates efficiently with any size of memory without being dependent on the memory size.

[0038] FIG. 6 is a flow diagram illustrating an ephemeral garbage collection process. The ephemeral garbage collection process may be initiated whenever additional memory is needed, upon a pre-determined interval, and the like. At block 602, a list of cards that have been accessed since the last garbage collection is obtained. This list is provided by the write-watch mechanism that is the subject of the afore-mentioned patent application. The present garbage collection process requests the list and stores the information that is returned from the write-watch mechanism in the write-watch information 408 shown in FIGURE 4. As mentioned above, the ephemeral garbage collection process sets the write-watch mechanism to track accesses to the card table(s). By monitoring the card table(s), the write-watch mechanism conveniently identifies the cards having objects that have been accessed and returns the identified cards in the list. Processing continues at block 604.

[0039] At block 604, the bundle table is updated based on the write-watch information. The bundle table is updated by marking each bundle that has one of its cards identified within the list. Marking of the bundle table may be performed in various ways. One technique is to set or clear a bit in a bitmap, where each bit is associated with a particular bundle. As illustrated in FIG. 4, each bundle is associated with a pre-determined number of cards. In certain cases, the last bundle may have fewer cards than

the previous bundles. In one implementation, the bundle may be the size of a page of card table memory. Processing continues at block 606.

[0040] At block 606, the bundle tables(s) are scanned to determine which bundles indicate that one of the cards associated with the bundle have been accessed. For each bundle that indicates that the bundle has been accessed, processing of "For" loop 608-609 is performed. Within "For" loop 608-609, at block 610, the card table is scanned to determine which cards in the bundle have been set. For any card in the bundle that is set, "For" loop 612-613 is performed.

[0041] In "For" loop 612-613, at block 614, each object in the card is scanned. The processing performed in block 614 may use conventional garbage collection processing utilized with card-marking. Once all the object have been identified in block 614, at block 616, the objects in the card that have been accessed may be collected in various manners that are known to those skilled in the art. For example, the accessed objects may be moved to the ephemeral generation or the like. Once each object in each card of each bundle that has been set has been processed, the ephemeral process 600 may reset the card table and the bundle table as needed.

[0042] Thus, as illustrated in FIG. 6, the present ephemeral garbage collection utilizes bundles without incurring the overhead of prior attempts. This allows the present ephemeral garbage collection to operate efficiently with any heap size, in particular, large heaps. For example, assuming a 4 Giga-byte (Gbyte) memory heap, in one implementation, the cards size may be set at 32 bytes. Because the size of the cards is relatively small, the time consumed for scanning the marked cards and looking for modified objects is not significant. For any cards that have modified objects, the card is marked by setting a bit within the card table. Therefore, each bit in the card table

represents 32 bytes of memory heap. Assuming a page is 4096 bytes, there will be 128 cards per page of memory (4096/32 = 128). These 128 cards will each be represented by one bit in the card table, which correlates to 16 bytes (128 bits). The bundle may then be set to encompass an entire page of card table memory, which will include 4096*8 cards.

[0043] The 4 giga-byte memory heap may be divided into 1,048,576 (i.e., 1M) worth of pages (4 Gbytes/4K). Thus, 128*1M cards covers the entire 4 Gbytes of memory heap. Thus, 16Mbytes ((16bytes/128 cards) * (128*1M cards)) of card table memory is utilized to cover the 4 Gbytes of memory heap. The write-watch mechanism may then be set to monitor the 16Mbytes of card table memory, which effectively tracks the entire 4Gbytes of memory.

[0044] By instructing the write-watch mechanism to track updates to the card table, the present ephemeral garbage collection process may utilize the existing code that performs the card-marking function for the garbage collection process without modifying or re-testing the code. In addition, the write-watch mechanism does not add additional costs to implement or adversely affect run-time performance for implementing bundles in the present ephemeral garbage collection process.

[0045] Reference has been made throughout this specification to "one embodiment," "an embodiment," or "an example embodiment" meaning that a particular described feature, structure, or characteristic is included in at least one embodiment of the present invention. Thus, usage of such phrases may refer to more than just one embodiment. Furthermore, the described features, structures, or characteristics may be combined in any suitable manner in one or more embodiments.

[0046] One skilled in the relevant art may recognize, however, that the invention may be practiced without one or more of the specific details, or with other

methods, resources, materials, *etc*. In other instances, well known structures, resources, or operations have not been shown or described in detail merely to avoid obscuring aspects of the invention.

[0047] While example embodiments and applications have been illustrated and described, it is to be understood that the invention is not limited to the precise configuration and resources described above. Various modifications, changes, and variations apparent to those skilled in the art may be made in the arrangement, operation, and details of the methods and systems of the present invention disclosed herein without departing from the scope of the claimed invention.